JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

SECTOR-128, NOIDA

DATA STRUCTURE PROJECT

Topic : MetroMate

SUBMITTED BY:		SUBMITTED TO:
Krishna Sahu	9923103098	Prof. Mukesh Saraswat
Sambhav Saxena	9923103103	
Jagriti Iyer	9923103113	
Ojaswi	9923103088	

Course Name: Data Structures Course Code: 15B17CI371 Program: B.Tech CSE

Index:

Introduction:

The MetroMate project is a comprehensive C++ application designed to provide users with a convenient and feature-rich solution for navigating metro systems. The app's primary functionality includes finding the shortest path between metro stations and identifying strongly connected stations.
Additionally, it offers an engaging Sudoku solver game for users to enjoy during their metro journeys.

Key Features ➔

Shortest Path Finder:

MetroMate features an efficient algorithm to calculate the shortest path between two metro stations.

Users can input their current station and destination, and the app will provide them with the optimal route, including transfers if necessary.

Real–time updates on estimated travel time and expected arrival at the destination are displayed.

Strongly Connected Stations:

The app identifies and highlights stations that have strong connections with multiple metro lines.

Users can explore these stations for easier transfers and access to multiple routes.

Interactive Metro Map:

MetroMate boasts an intuitive and user–friendly metro map interface.

Users can interact with the map to explore the metro system and plan their journeys.

Sudoku Solver Game:

To keep users entertained during their metro rides, the app offers a Sudoku solver game.

Users can input Sudoku puzzles, and the app will solve them or generate new puzzles for endless entertainment.

User Profiles and History:

MetroMate allows users to create profiles and store their travel history.

Users can save their favorite routes and frequently visited stations for quick access.

User-Friendly Interface:

The app features an intuitive and aesthetically pleasing design, ensuring a seamless user experience.

Clear icons, easy navigation, and informative tooltips make it accessible to users of all ages.

Real-time Updates:

Users receive real–time information on metro schedules, delays, and any service disruptions.

Notifications and alerts help users stay informed about their metro system's status.

MetroMate is an all–in–one metro station app designed to simplify metro navigation and offer entertainment to users during their commutes. With its robust features, intuitive interface, and future development potential, MetroMate aims to become the go-to app for metro travelers seeking convenience and enjoyment on their journeys.

Concepts used ➔

1) Graph
2) Hashing(unordered map)
3) BFS
4) Shortest path algorithm
5) Backtracking(sudoku solver)
6) Queue
7) Vector
8) List

Below is a detailed description of the main components and functionalities of the code:

1. Graph Representation:

● The code includes a graph class that utilizes an adjacency list to represent connections between metro stations. Stations are nodes in the graph, and connections between them are edges. The class supports the addition of edges means connection of stations.

2. Graph Traversal and Shortest Path:

● The BFS (Breadth−First Search) algorithm is implemented to traverse the metro graph. It allows users to select a starting station and a destination station. The shortest path between these stations is then calculated using BFS. The program displays both the path and fare for the journey.

3. Sudoku Solver (Backtracking):

● The code features a Sudoku solver using the backtracking algorithm. Users can play a Sudoku puzzle, and the program provides the solution by filling in the empty cells. The solver employs a recursive approach, attempting different numbers at each cell until a valid solution is found.

4. User Authentication System:

● User authentication is implemented using a simple Users class. The program prompts users to log in by entering a username and password..

5. Main Menu and User Interaction:

● The main function has a user−friendly menu that allows users to choose from various options:

● Play Sudoku: Users can interactively play Sudoku, and the program provides the solution.

● Login: Users can log in with a username and password .

● Signup: A feature to sign up for a new account .

● Book a Ticket: Users can book metro tickets after logging in.

● User history: user metro travel history is stored and displayed.

● Check Shortest Route: Users can find the shortest route between metro stations and determine the corresponding ticket price.

6. STL Containers:

● The code utilizes Standard Template Library (STL) containers, including vector, queue, and unordered_map for efficient data storage and manipulation

SOURCE CODE ➔

```
#include<iostream>
#include<vector>
#include<queue>
#include<unordered_map>
#include<algorithm>
#include<list>

#define cost 9 // the cost
#define N 9 // for soduku

using namespace std;

/*

MAP

*/
template <class T>
class graph{

vector<T> ans;
unordered_map<T,bool> visited;
unordered_map<T,T> parent;

public:
unordered_map<T,list<T>> adj;

void addEdge(T u,T v,bool direcTIon){
adj[u].push_back(v);

if(!direcTIon){
adj[v].push_back(u);
```

```
}
}

void print_adj(){
for(auto i:adj){
cout<<i.first<<" -> ";
for(auto j:i.second){
if(j==i.second.back()) cout<<j<<" ";
else cout<<j<<" , ";
}
cout<<endl;
}
}

void bfs(unordered_map<T,bool> &visited,T node,vector<T> &ans){
queue<T> q;

q.push(node);
visited[node] = true;

while(!q.empty()){
T front_node = q.front();
q.pop();

// put it in ans
ans.push_back(front_node);
for(auto i: adj[front_node]){

if(!visited[i]){
q.push(i);
visited[i] = true;
}

}
```

```
}
}
vector<T> BFS(T node){
vector<T> ans;
unordered_map<T,bool> visited;

bfs(visited,node,ans);
return ans;
}

vector<T> shortest_path(T src, T dest){
queue<T> q;
q.push(src);
parent[src] = src;
visited[src] = true;

while(!q.empty()){
T front = q.front();
q.pop();

for(auto i:adj[front]){
if(!visited[i]){
visited[i] = true;
parent[i] = front;
q.push(i);
}
}
}

T current = dest;
ans.push_back(current);

while(current!=src){
current = parent[current];
ans.push_back(current);
```

```
}

reverse(ans.begin(),ans.end());
return ans;
}
};

graph<string> g;

class user {
public:
string username;
string password;
vector<string> path;

vector<vector<string>> history;

void book() {
cout << "\n Enter boarding station: " << endl;
string boar;
cin >> boar;

cout << "\n Final destination station: " << endl;
string dest;
cin >> dest;

// Clear the exisTIng path before storing the new one
path.clear();

// Get the shortest path from the boarding station to the destination
path = g.shortest_path(boar, dest);

// Check if the path is not empty before storing and prinTIng
if (!path.empty()) {
cout << "Path stored: ";
```

```
for (const auto &station : path) {
cout << station << " ";
}
cout << endl;

// Store the path in the user's history
history.push_back(path);
} else {
cout << "No valid path found from " << boar << " to " << dest << endl;
}
}

void show(){
for(int i=0;i<history.size();i++){
for(int j=0;j<history[i].size();j++){
cout<<history[i][j]<<" ";
}
}
}
};

user users[6900];
int cnt = 0;

void signup(string username, string pass) {
users[cnt].username = username;
users[cnt].password = pass;

/*if (username.length() < 3 || pass.length() < 3) {
cout << "At least 3 characters are required!\n";
return;
}*/

cnt++;
cout<<" Successfully signup ! "<<endl;
```

```
}

user userlogin(string username, string pass) {
for (auto &us : users) {
if (us.username == username && us.password == pass) {
return us;

}
}

cout << "No such user found!" << endl;
return user();
}

void metro(){

string src1;
cout<<"\n Enter here: "<<endl;
cin>>src1;
cout<<" Select boarding StaTIon : "<<endl;
vector<string> stations = g.BFS(src1);
for(auto i:stations){
cout<<i<<" ";
}

string src;
cout<<"\n Enter here: "<<endl;
cin>>src;

bool okj = false;

for(auto i: stations){
if(i==src){
okj = true;
}
```

```
}

if(!okj){
cout<<" You entered wrong station ! "<<endl;
return;
}

cout<<"Select Destination Station: "<<endl;

for(auto i:stations){
cout<<i<<" ";
}

string dest;
cout<<"\n Enter here: "<<endl;
cin>>dest;

okj = false;

for(auto i: stations){
if(i==dest){
okj = true;
}
}

if(!okj){
cout<<" You entered wrong station ! "<<endl;
return;
}

vector<string> path = g.shortest_path(src,dest);

cout<<"\n\n The path from "<<src<<" to "<<dest<<" is ->> ";
for(auto i:path){
if(i==path.back()) cout<<i<<" ";
```

```
else cout<<i<<" - ";
}
cout<<endl;

cout<<" \n The fare for the ticket is : ";
int ticketprice = (path.size()-1) * cost;
cout<<ticketprice<<endl;

}

bool searchLocation(int puzzle[N][N], int& row, int& col);
bool isValid(int puzzle[N][N], int row, int col, int number);

bool SolveSudoku(int puzzle[N][N])
{
int row, col;
if (!searchLocation(puzzle, row, col))
return true;

for (int number = 1; number <= 9; number++)
{
if (isValid(puzzle, row, col, number))

{
puzzle[row][col] = number;
if (SolveSudoku(puzzle))
return true;
puzzle[row][col] = 0;
}
}
return false;
}
bool searchLocation(int puzzle[N][N], int& row, int& col)
{
for (row = 0; row < N; row++)
for (col = 0; col < N; col++)
if (puzzle[row][col] == 0)
return true;
return false;
}
bool uRow(int puzzle[N][N], int row, int number)
{
for (int col = 0; col < N; col++)
if (puzzle[row][col] == number)
return true;
return false;
}
bool uCol(int puzzle[N][N], int col, int number)
{
for (int row = 0; row < N; row++)
if (puzzle[row][col] == number)
return true;
return false;
}

bool uBox(int puzzle[N][N], int bsRow,
int bsCol, int number)
{
for (int row = 0; row < 3; row++)
for (int col = 0; col < 3; col++)
if (puzzle[row + bsRow][col + bsCol] == number)
return true;
return false;
}
bool isValid(int puzzle[N][N], int row, int col, int number)
{
return !uRow(puzzle, row, number)
&& !uCol(puzzle, col, number)
&& !uBox(puzzle, row - row % 3,
col - col % 3, number)
```

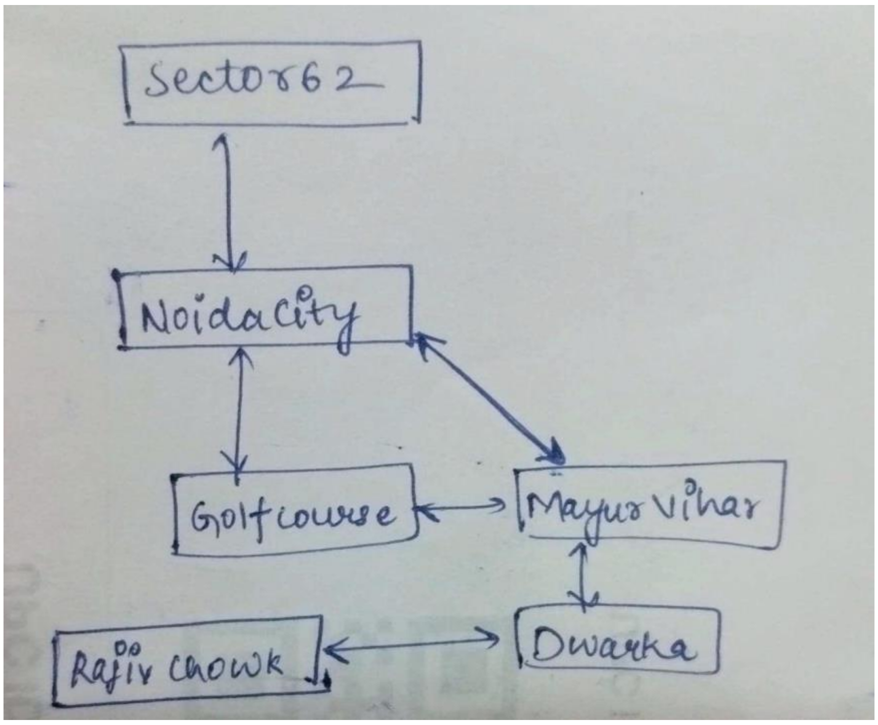
```
&& puzzle[row][col] == 0;
}
void printSudoku(int puzzle[N][N])
{
for (int row = 0; row < N; row++)
{
for (int col = 0; col < N; col++)
cout << puzzle[row][col] << " ";
cout << endl;
}
}

void play(){
int puzzle[N][N] = { { 1, 0, 6, 0, 0, 2, 3, 0, 0 },
{ 0, 5, 0, 0, 0, 6, 0, 9, 1 },
{ 0, 0, 9, 5, 0, 1, 4, 6, 2 },

{ 0, 3, 7, 9, 0, 5, 0, 0, 0 },
{ 5, 8, 1, 0, 2, 7, 9, 0, 0 },
{ 0, 0, 0, 4, 0, 8, 1, 5, 7 },
{ 0, 0, 0, 2, 6, 0, 5, 4, 0 },
{ 0, 0, 4, 1, 5, 0, 6, 0, 9 },
{ 9, 0, 0, 8, 7, 4, 2, 1, 0 } };
cout << "You can play puzzle " << endl;
printSudoku(puzzle);

cout<<"Enter any number to see solutions \n\n"<<endl;
int xi;

cin>>xi;
cout<<"\n\n";
SolveSudoku(puzzle);
cout<<"Solution of the puzzle "<<endl;
printSudoku(puzzle);
}
```

```
int main(){

cout<<" ------------------------------------------METRO PROJECT----------
--------------------------------\n\n"<<endl;
cout<<" Main Menu and graph"<<endl;
cout<<" Login System , Synopsis "<<endl;
cout<<" Soduku Solver \n\n"<<endl;

cout<<"*************** setting up our metro mate ********* "<<endl;

// taking input

cout<<"Enter data: (first line - number of connections n then n lines -
connections ) "<<endl;
int n; // number of edges
cin>>n;
string u, v; // edges
for(int i=0;i<n;i++){
cin>>u>>v;
g.addEdge(u,v,0);
}

/* dummy data to input -

6
noidacity golfcourse
noidacity mayurvihar
mayurvihar golfcourse
mayurvihar dwarka
dwarka rajivchok
sector62 noidacity

*/
cout<<"\n\n Printing Adjacency List of Stations of BLUE LINE ==>>
\n\n"<<endl;
g.print_adj();

bool login = false;

user k;

while(true){
cout<<"\n\n\n-----------------MAIN MENU-----------------"<<endl;
cout<<"1) Play and solve Soduku "<<endl;
cout<<"2) Login "<<endl;
cout<<"3) Signup "<<endl;
cout<<"4) Book a TIcket "<<endl;
cout<<"5) Check shortest route to reach destination and check price for
it"<<endl;
cout<<"6) Show user history "<<endl;
cout<<"7) Exit"<<endl;
int ch;
cout<<"Enter choice: ";
cin>>ch;

if(ch==1){
play();
}
else if(ch==2){
if(!login){
cout<<"Enter username: "<<endl;
string username;
cin>>username;

cout<<"Password: "<<endl;
string password;
cin>>password;
```

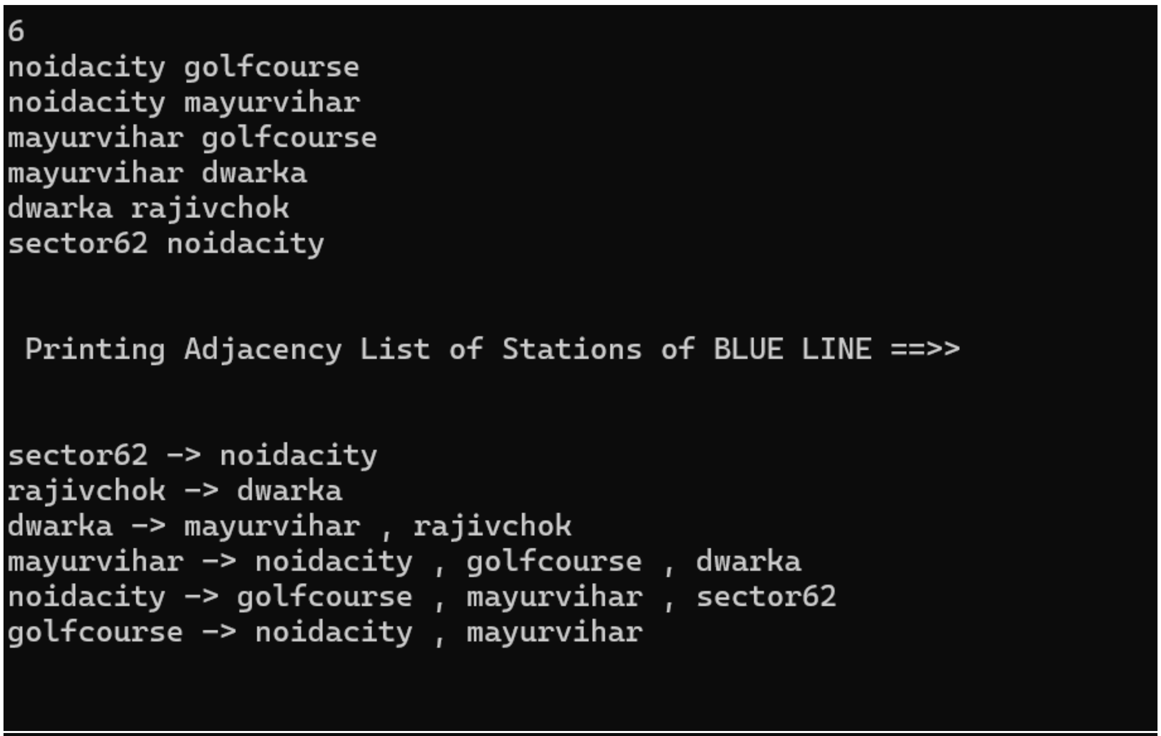
```
k = userlogin(username,password);
cout<<"Hello "<<k.username<<endl;

login = true;
}
else{
cout<<" You are already login ! "<<endl;
}
}
else if(ch==3){
cout<<"\n\n Enter username : ";
string user;
cin>>user;
cout<<"Enter password: "<<endl;
string pass;
cin>>pass;
signup(user,pass);
}
else if(ch==4){

if(!login){
cout<<"\n You need to login first in order to book the Tickets !
\n"<<endl;
}
else{
k.book();
}

}
else if(ch==5){
metro();
}
else if(ch==6){
k.show();
}
```

```
else if(ch==7){
break;
}
else{
cout<<"\n\n WRONG CHOICE "<<endl;
}

}
cout<<"\n\n------ THANKS FOR USING METRO MATE -------"<<endl;}
```

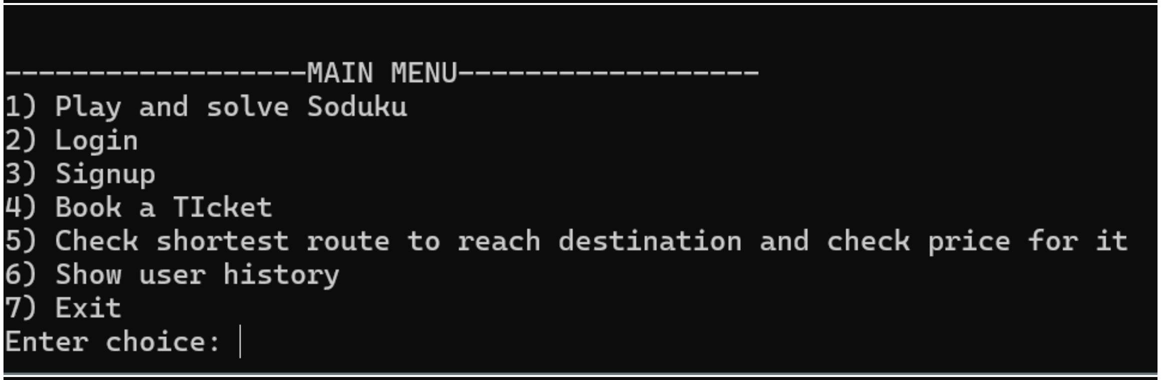
GRAPH REPRESENTATION ➔



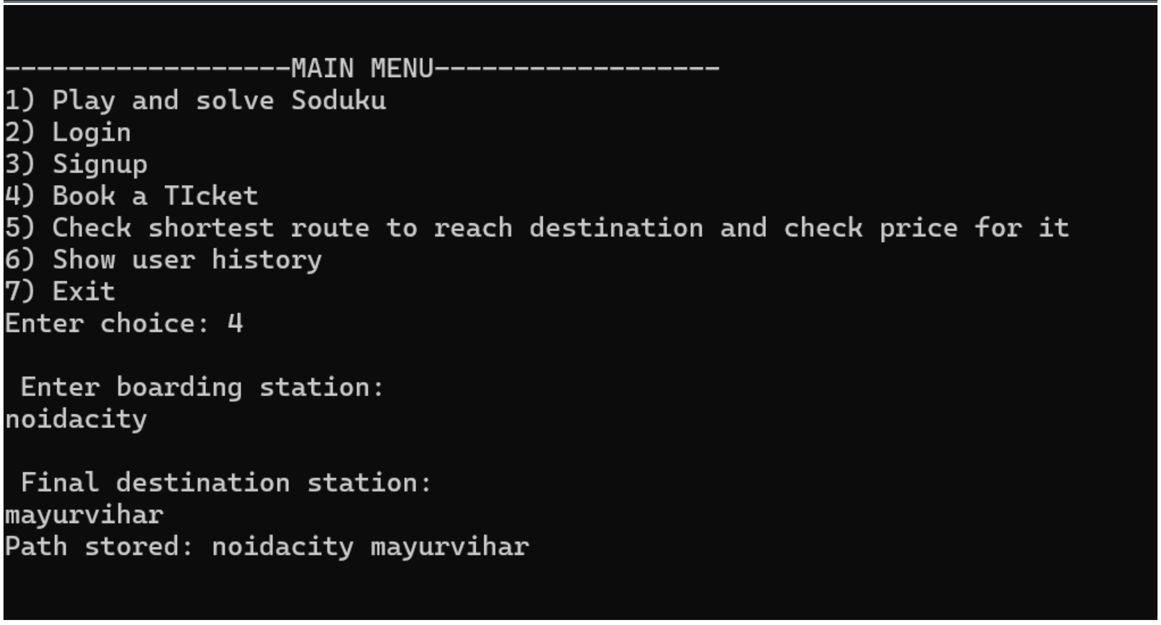
OUTPUTS ->



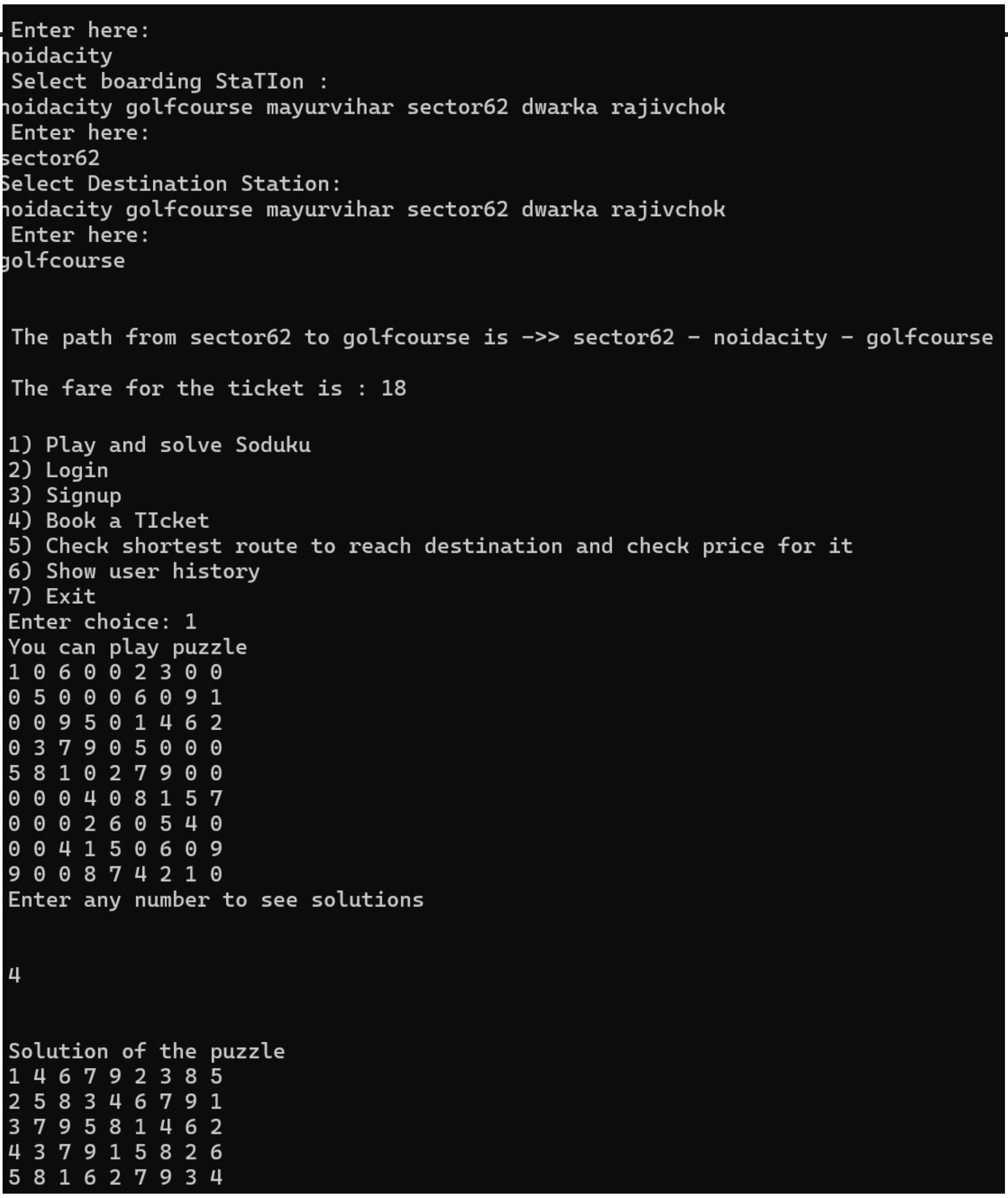
```
6
noidacity golfcourse
noidacity mayurvihar
mayurvihar golfcourse
mayurvihar dwarka
dwarka rajivchok
sector62 noidacity


 Printing Adjacency List of Stations of BLUE LINE ==>>


sector62 -> noidacity
rajivchok -> dwarka
dwarka -> mayurvihar , rajivchok
mayurvihar -> noidacity , golfcourse , dwarka
noidacity -> golfcourse , mayurvihar , sector62
golfcourse -> noidacity , mayurvihar
```



```
------------------MAIN MENU------------------
1) Play and solve Soduku
2) Login
3) Signup
4) Book a TIcket
5) Check shortest route to reach destination and check price for it
6) Show user history
7) Exit
Enter choice:
```



```
------------------MAIN MENU------------------
1) Play and solve Soduku
2) Login
3) Signup
4) Book a TIcket
5) Check shortest route to reach destination and check price for it
6) Show user history
7) Exit
Enter choice: 4

 Enter boarding station:
noidacity

 Final destination station:
mayurvihar
Path stored: noidacity mayurvihar
```



```
 Enter here:
noidacity
 Select boarding StaTIon :
noidacity golfcourse mayurvihar sector62 dwarka rajivchok
 Enter here:
sector62
Select Destination Station:
noidacity golfcourse mayurvihar sector62 dwarka rajivchok
 Enter here:
golfcourse


 The path from sector62 to golfcourse is ->> sector62 - noidacity - golfcourse

 The fare for the ticket is : 18

1) Play and solve Soduku
2) Login
3) Signup
4) Book a TIcket
5) Check shortest route to reach destination and check price for it
6) Show user history
7) Exit
Enter choice: 1
You can play puzzle
1 0 6 0 0 2 3 0 0
0 5 0 0 0 6 0 9 1
0 0 9 5 0 1 4 6 2
0 3 7 9 0 5 0 0 0
5 8 1 0 2 7 9 0 0
0 0 0 4 0 8 1 5 7
0 0 0 2 6 0 5 4 0
0 0 4 1 5 0 6 0 9
9 0 0 8 7 4 2 1 0
Enter any number to see solutions


4


Solution of the puzzle
1 4 6 7 9 2 3 8 5
2 5 8 3 4 6 7 9 1
3 7 9 5 8 1 4 6 2
4 3 7 9 1 5 8 2 6
5 8 1 6 2 7 9 3 4
```

Project Outcomes->

Programming Skills:

1. Graph Theory and Algorithms:
    - Graph Representation: Understanding how to represent and manipulate graphs using adjacency lists.
    - Breadth-First Search (BFS): Implementing BFS to find the shortest path in an unweighted graph.
    - Pathfinding: Using BFS to determine the shortest route between two nodes.
2. Data Structures:
    - Queues: Utilizing queues for BFS implementation.
    - Hash Maps: Using unordered maps to store adjacency lists, visited nodes, and parent relationships.
    - Vectors: Managing dynamic arrays for storing paths and results.
3. User Management:
    - Signup/Login System: Implementing a basic user authentication system.
    - Session Management: Handling user sessions and storing user-specific data like travel history.
4. Problem Solving:
    - Backtracking Algorithm: Implementing a backtracking algorithm to solve Sudoku puzzles.
    - Validation Functions: Writing functions to validate user input and Sudoku puzzle constraints.